

Start with fib. Why is the recursive definition such a poor performer.
Better: fill in an array.

You recall dynamic programming I hope:

Solve small solutions.
Glue them together.
Let's look at the classic: string comparison

some vs. sammy

Draw the matrix. Figure out the rule for going backwards
work back towards the beginning.

Sometimes can improve on things -- Ukkonen trick

Michael Trick notes on the website shows the use of dynamic programming
for NP-complete problems.

Duellists (dynamic programming applied to game playing; on your own):

```
/ duel.k, Dennis Shasha, October 2001
/ Alternate moves.
/ Player 1 goes first at multiples of 20 (or in our case, even distances).
/ Duellists -- Keith Carradine and Harvey Keitel
/ Napoleon Pistol. Gribeauval 1806. flintlock.
/ One doesn't hear about duels so often these days, so Ecco was
/ quite surprised to hear the request from one Austerlitz Toulemonde.
/ "I am from the Napoleonic Society," he explained.
/ "We use firearms from the time of the Emperor and when disagreements arrive
/ we allow duels, though only to the first touch and
/ our vests are made of kevlar.
/ Even when we're wounded, no one gets too badly hurt,
/ because our members prefer modern surgeons
/ to barbers.
/
/ Now it happens that I have been caught in {\em flagrant delit}
/ with the extremely charming wife of another member.
/ To protect his honor, he has slapped me on the face and
/ we must duel tomorrow.
/ In this duel, I am the offender and he is the challenger.

/ "The basic duel scenario here is that there are two people
/ start by standing 100 paces apart.
/ The offender has two bullets and the challenger has three
/ bullets.
/ So, he has more bullets.
/ At each distance, starting with the offender,
/ one duellist can choose to shoot at most one bullet
/ provided he has any left.
/ Then the other duellist can choose to shoot
/ at most one bullet if he has any left.
/ If neither is hit at that distance, each dueler walks 5 paces towards
/ the other, thus bringing the duelers 10 paces closer to one another.
/ The duellist who went first the last time goes second this time.

/ "Now the probability that a dueler will hit his opponent
/ grows as the distance shrinks, i.e.  $(100 - \text{dist})/100$ ."
```

```

/ "So if either of you has a bullet left and the other has none, the
/ one with the bullet surely win," Liane said.
/ My question to you is: what should my strategy be?
/ Assuming that he

/ The duelers
/ A dueler is allowed to shoot only once at a certain distance and
/ the duelers approach one another 10 paces at a time (5 paces each)
/ after both each has a chance to shoot or not (and assuming neither is shot).
/ One's chance of touching is defined by the formula:
/ (maxdist-dist)/maxdist
/ For the purposes of this program, I've replaced 10 paces by 1 pace.

/ Now, we have glmat which is the best prob that player 1 can do from
/ given configuration (player 1's move, so much distance, so many
/ shots for player 1 and so many shots for player 2).
/ g2mat is the least winning prob for player 1 that player 2 can manage
/ when it is player 2's move.

/ To figure out the moves, find the first time that some player shoots and
/ then continue from that configuration.
/ All other configurations are simply unattainable assuming rational play.

/ To play this as a game, form the array but ignore the output from out.
/ Then use doesshoot1 or doesshoot2 to decide which is the best move.

/ Start with glmat.

/ APPLICATION SPECIFIC

/ prob of hit and miss
p: {[dist] (maxdist - dist) % maxdist}

q: {[dist] 1 - p[dist]}

/ whatever the distance probability player 1 has a slightly better shot
improve: 1 / if improve: 1 then both have same prob of hitting at distance.
p1: {[dist] improve*p[dist]}
q1: {[dist] 1 - p1[dist]}

p2: p
q2: q

/ probability that player 1 will win if he has m1 bullets
/ and m2 has m2 bullets and they are dist apart
/ and it is player 1's turn and whether player 1 went first or not
/ is determined by the isfirst flag
gl: {[dist; m1; m2; isfirst]
    if[glmat[dist;m1;m2; isfirst] > -1; :glmat[dist;m1;m2; isfirst]]
    / greater than -1 means already set
    / :(g2[dist;m1;m2]) | ((p[dist]) + (q[dist] * g2[dist;m1-1;m2]))
    if[isfirst = 1
        x1: (g2[dist;m1;m2; 0]) / player 1 doesn't shoot
        x2: ((p1[dist]) + (q1[dist] * g2[dist;m1-1;m2; 0]))
        / player 1 shoots and succeeds with prob p1[dist] or misses and then
        / prob depends on the g2.
    ]

```

```

    if[isfirst = 0 / if player 1 was not first this time he will be next time
      / so recursion goes to player 1
      x1: (g1[dist-1;m1;m2; 1]) / you try at a smaller distance
      x2: ((p1[dist]) + (q1[dist] * g1[dist-1;m1-1;m2; 1])) / you shoot
      / at this distance and then you get another chance at a smaller
      / distance because the chance to go first alternates.
      / never take this second choice
    ]
    if[x2 > x1 / shoots
      if[m1 > 1
        out,: ,("Player 1 at "), ($dist), (" having "), ($m1), (" bullets, shoots.
Prob to win: "), ($x2), (" Player 2 has: "), ($m2)
        out,: :[isfirst; (" Player 1 was first."); ("Player 1 was second.")]
      ]
      if[m1 = 1
        out,: ,("Player 1 at "), ($dist), (" having one bullet, shoots. Prob to win:
"), ($x2), (" Player 2 has: "), ($m2)
        out,: :[isfirst; (" Player 1 was first."); ("Player 1 was second.")]
      ]
    ]
    :x1 | x2
    / first part is if player 1 doesn't shoot.
    / second part is if player 1 does.
  }

/ probability that player 1 will win if he has m1 bullets
/ and m2 has m2 bullets and they are dist apart
/ and it is player 2's turn and whether player 2 is going first at this
/ distance is determined by the isfirst flag
g2:{{dist; m1; m2; isfirst}
  / if[m1 = 0; :0]
  / if[m2 = 0; :1]
  if[g2mat[dist;m1;m2; isfirst] > -1; :g2mat[dist;m1;m2; isfirst]]
    / -1 is the initial value; so if not that then this is known
  if[1 = isfirst
    x1:(g1[dist;m1;m2;0]) / if player 2 chooses not to shoot,
    / control passes to player 1
    x2: (q2[dist] * g1[dist;m1;m2-1;0]) / if player 2 hits, then
    / value to player 1s 0; else with prob q2[dist], prob of
    / player 1 winning is g1[dist;m1;m2-1;0]
  ]
  if[0 = isfirst / if player 2 is not first this time, he will be next time
    x1:(g2[dist-1;m1;m2;1]) / if second this time, then be first the next time
    x2: (q2[dist] * g2[dist-1;m1;m2-1;1]) / or try to shoot and remaining
    / value is prob of missing * prob player 1 wins if player 2 goes
    / first next
  ]
  if[x2 < x1 / note that player 2 is a minimizer
    if[m2 > 1
      out,: ,("Player 2 at "), ($dist), (" having "), ($m2), (" bullets, shoots.
Prob to win: "), ($1-x2), (" Player 1 has: "), ($m1)
      out,: :[isfirst; (" Player 2 was first."); ("Player 2 was second.")]
    ]
    if[m2 = 1
      out,: ,("Player 2 at "), ($dist), (" having one bullet, shoots. Prob to win:
"), ($1-x2), (" Player 1 has: "), ($m1)
      out,: :[isfirst; (" Player 2 was first."); ("Player 2 was second.")]
    ]
  ]
  :x1 & x2
  / first part is player 2 not shooting so distance gets closer by 10

```

```

/   paces (here represented as 1)
/ second part is player 2 shooting.
}

/ Once we've constructed the matrix, we answer the question about whether
/ we shoot or not at a certain distance.
doesshoot1: {[dist; m1;m2; isfirst]
  if[isfirst = 0; :0] / never worthwhile to shoot in that case
  if[m1 = 0; :0] / can't shoot if have nothing
  if[isfirst = 1
    x1: (g2mat[dist;m1;m2;0])
    x2: ((p1[dist]) + (q1[dist] * g2mat[dist;m1-1;m2;0]))
  ]
  :~ x2 < x1
}

doesshoot2: {[dist; m1;m2; isfirst]
  if[isfirst = 0; :0] / never worthwhile to shoot in that case
  if[m2 = 0; :0] / can't shoot if have nothing
  if[isfirst = 1
    x1: (g1mat[dist;m1;m2;0])
    x2: (q2[dist] * g1mat[dist;m1;m2-1;0])
  ]
  :~ x2 < x1
}

/ DATA

out: ()
maxdist: 10 / maxdist sets of 10 paces
/ set up
` 0: "Which player number are you? (1, 2)\n"
x: 0: `
zz: : [("1") _in x; playernum: 1; playernum: 2]
` 0: "How many bullets does player 1 have? (0... 5)\n"
x: 0 $ 0: `
m1: x
` 0: "How many bullets does player 2 have? (0... 5)\n"
x: 0 $ 0: `
m2: x

num1: m1 / first mover bullets
num2: m2 / second mover bullets

/ for every distance, for each number of bullets left for each player
/ fourth entry is whether a certain player is first or not.
g1mat: ((maxdist+1); (num1+1); (num2+1); 2) # -1
      / prob that player 1 wins -- player 1 moves
g2mat: ((maxdist+1); (num1+1); (num2+1); 2) # -1
      / prob that player 1 wins -- player 2 moves

/ initialization

g1mat[0;;;]: 1 / if players are at 0 distance away, player 1 will win if he
              / has bullets; first entry is distance.
g2mat[0;;;]: 0 / if players are at 0 distance away, player 1 will lose if
              / player 2 has bullets; first entry is distance. player 2 is a
              / minimizer.
g1mat[;;0]: 1 / if player 2 ever has 0 bullets, player 1 is sure to win
              / provided he has bullets; third entry is player 2 bullets

```

```
g1mat[;0;;]: 0 / if player 1 has 0 bullets, he will lose. Not sure needed
              / second entry is player 1 bullets.
```

```
g2mat[;0;;]: 0 / if player 1 ever has 0 bullets, player 1 is sure to lose
g2mat[;;0;]: 1 / if player 2 ever has 0 bullets, player 1 wins. Needed?
```

```
g1mat[0;0;0;]: 52 / neutral, neither player wins but can't get here
g2mat[0;0;0;]: 52 / neutral, neither player wins but can't get here
```

```
/ EXECUTION
```

```
idist: 0
while[idist < (maxdist+1)
  m1: 0
  while[m1 < (num1+1)
    m2: 0
    while[m2 < (num2+1)
      g1mat[idist;m1;m2;0]: g1[idist; m1; m2;0]
      g2mat[idist;m1;m2;0]: g2[idist; m1; m2;0]
      g1mat[idist;m1;m2;1]: g1[idist; m1; m2;1]
      g2mat[idist;m1;m2;1]: g2[idist; m1; m2;1]
      m2+: 1
    ]
    m1+: 1
  ]
  idist+: 1
]
```

```
/ ` 0: ?out
```

```
even: {[x] (_ x % 2) = (x % 2)}
```

```
out2: ()
idist: maxdist
m1: num1
m2: num2
```

```
while[idist > 0
  onefirst: even[idist]
  if[1 = onefirst / player 1 goes first
    if[m1 > 0
      if[doesshoot1[idist;m1;m2;1]
        out2,: ,("Player 1 (going first) at "), ($idist), (" having "), ($m1), ("
bullet(s), shoots. Player 2 has: "), ($m2), (" Prob 1 wins: "),
($g1mat[idist;m1;m2;1])
        m1-: 1
      ]
    ]
    if[(m2 > 0)
      if[doesshoot2[idist;m1;m2;0]
        out2,: ,("Player 2 (going second) at "), ($idist), (" having "), ($m2), ("
bullet(s), shoots. Player 1 has: "), ($m1), (" Prob 2 wins: "), ($1-
g2mat[idist;m1;m2;0])
        m2-: 1
      ]
    ]
  ]
  if[0 = onefirst / player 1 goes second
    if[(m2 > 0)
      if[doesshoot2[idist;m1;m2;1]
```

```

        out2, := ("Player 2 (going first) at "), ($idist), (" having "), ($m2), ("
bullet(s), shoots. Player 1 has: "), ($m1), (" Prob 2 wins: "), ($1-
g2mat[idist;m1;m2;1])
        m2-:= 1
    ]
]
if[m1 > 0
    if[doesshoot1[idist;m1;m2;0]
        out2, := ("Player 1 (going second) at "), ($idist), (" having "), ($m1), ("
bullet(s), shoots. Player 2 has: "), ($m2), (" Prob 1 wins: "),
($g1mat[idist;m1;m2;0])
        m1-:= 1
    ]
]
]
idist-:= 1
]
/ definitely shoot at 0 distance
if[m1 > 0
    out2, := ("Player 1 (going first) at "), ($idist), (" having "), ($m1), ("
bullet(s), shoots. Player 2 has: "), ($m2), (" Prob 1 wins: "),
($g1mat[idist;m1;m2;1])
    m1-:= 1
]
if[m2 > 0
    out2, := ("Player 2 (going second) at "), ($idist), (" having "), ($m2), ("
bullet(s), shoots. Player 1 has: "), ($m1), (" Prob 2 wins: "), ($1-
g2mat[idist;m1;m2;0])
    m2-:= 1
]
` 0: out2

/ set up
file: "duelshare"
if[0 < # _i
    file: _i[0]
]

parse: {[lines; orig1; orig2]
    m1: orig1
    m2: orig2
    distarr: ()
    k: 0
    while[k < #lines
        line: lines[k]
        flag: (0 = # & ~ line = " ") | (0 = #line)
        if[~ flag / non-empty
            i: line ? ", "
            dist: 0 $ line[!i]
            distarr, := dist
            line: (i+1) _ line
            i: line ? ", "
            player: 0 $ line[!i]
            line: (i+1) _ line
            shot: (line _sm "*sh*") | (line _sm "*sh*")
            if[shot
                :[player=1; m1-:= 1; m2-:= 1]
            ]
        ]
    ]
    k+: 1
}

```

```

]
if[0 = #distarr
    :(m1; m2; 10)
]
if[1 = #distarr
    :(m1; m2; _ dist % 10)
]
:[dist = distarr[(#distarr)-2] / same distance twice
    :(m1; m2; _ (dist-10) % 10)
    :(m1; m2; _ dist % 10)]
}

m1: num1
m2: num2
` 0: "Shall we play? (y,n)\n"
x: 0: `
while[(x _sm "*y*") | (x _sm "*Y*")
    a: 0: file
    triple: parse[a; num1; num2]
    m1: triple[0]
    m2: triple[1]
    r: triple[2]
    if[(playernum = 1)
        ishoot: 0
        if[(even[r])
            ishoot: doesshoot1[r;m1;m2;1]
        ]
        if[ishoot
            a,: ,($10*r), (","), ($playernum), (" shoot")
            / ` 0: ("Player 1 shoots at "),( $10*r), (" with "), ($m1), (" bullets
vs. "),( $m2), (" bullets for player 2.\n")
        ]
        if[~ ishoot
            a,: ,($10*r), (","), ($playernum), (" pass")
        ]
    ]
    if[(playernum = 2)
        ishoot: 0
        if[~ (even[r])
            ishoot: doesshoot2[r;m1;m2;1]
        ]
        if[ishoot
            a,: ,($10*r), (","), ($playernum), (" shoot")
            / ` 0: ("Player 2 shoots at "),( $10*r), (" with "), ($m2), (" bullets
vs. "),( $m1), (" bullets for player 1.\n")
        ]
        if[~ ishoot
            a,: ,($10*r), (","), ($playernum), (" pass")
        ]
    ]
    ` 0: a
    file 0: a
    ` 0: "Shall we play? (y,n)\n"
    x: 0: `
]

```